VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV POČÍTAČOVÉ GRAFIKY A MULTIMÉDIÍ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF COMPUTER GRAPHICS AND MULTIMEDIA

ZOBRAZENÍ 3D SCÉNY VE WEBOVÉM PROHLÍŽEČI

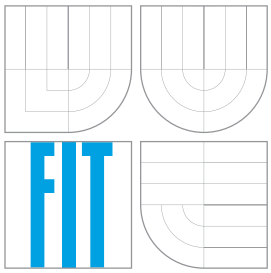
BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE
AUTHOR

TOMÁŠ SYCHRA

BRNO 2011

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY



FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV POČÍTAČOVÉ GRAFIKY A MULTIMÉDIÍ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF COMPUTER GRAPHICS AND MULTIMEDIA

ZOBRAZENÍ 3D SCÉNY VE WEBOVÉM PROHLÍŽEČI

DISPLAYING 3D GRAPHICS IN WEB BROWSER

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE
AUTHOR

TOMÁŠ SYCHRA

VEDOUCÍ PRÁCE
SUPERVISOR

Ing. MICHAL ŠPANĚL, Ph.D.

BRNO 2011

Abstrakt

Tato práce diskutuje aktuální možnosti zobrazení akcelerované 3D scény ve webovém prohlížeči a krátce shrnuje technologie pro zobrazení medicínských dat. V druhé části práce je navržena a implementována aplikace, která umožňuje prohlížení volumetrických medicínských dat přímo v okně prohlížeče, bez nutnosti instalace. Aplikace je postavena na technologiích WebGL, Javascript a grafickém enginu O3D API.

Abstract

This bachelor's thesis deals with display of accelerated 3D graphics in a web browser environment. Existing technologies such as WebGL are presented and discussed. Further, in the second part of the thesis, an application for browsing medical volumetric data is designed and implemented. The application is built with the WebGL technology and Javascript graphics engine called O3D API.

Klíčová slova

multiplanární zobrazeni, 3D grafika, akcelerovaná grafika, webový prohlížeč, medicínská data, ct, mri, volume rendering, objemová data

Keywords

multiplanar views, 3D graphics, accelerated graphics, web browser, medical data, ct, mri, volume rendering, volumetrics data

Citace

Tomáš Sychra: Zobrazení 3D scény ve webovém prohlížeči, bakalářská práce, Brno, FIT VUT v Brně, 2011

Zobrazení 3D scény ve webovém prohlížeči

Prohlášení

Prohlašuji, že jsem tuto bakalářskou práci vypracoval samostatně pod vedením pana Ing. Michala Španěla, Ph.D.

.....
Tomáš Sychra
13. května 2011

Poděkování

Tímto bych chtěl poděkovat vedoucímu své bakalářské práce, Ing. Michalu Španělovi Ph.D., za pomoc a čas který mi věnoval. V neposlední řadě bych rád poděkoval i svým rodičům za jejich velkou podporu.

Obsah

Kapitola 1

Úvod

Vývoj informačních technologií jde kupředu mílovými kroky. V poslední době dochází k velkému rozvoji pokročilých webových aplikací. Už se nejedná pouze o statické stránky, rozsáhlé databáze či dynamické stránky. V dnešní době vznikají zcela reálné aplikace. Příkladem může být velmi pokročilé online psaní dokumentů (Google Docs) od společnosti Google. Delší dobu řešili společnosti problém, jak zobrazovat 3D grafiku v okně webového prohlížeče. Bylo napsáno mnoho pluginů, které umožňují programovat akcelerovanou grafiku. Tyto pluginy bylo ale nutné explicitně doinstalovávat, což není příliš vhodné pro běžné uživatele. Akcelerovaná grafika je však zcela nepostradatelná, jelikož ani nejrychleší procesory nejsou schopny počítat pokročilou grafiku. Na scénu proto přišel zcela nový standard WebGL.

V rámci své bakalářské práce se budu věnovat právě těmto problémům. Za cíl jsem si zvolil vytvořit aplikaci pro multiplanární zobrazení medicínských dat. Aplikace bude navržena tak, aby uživatel nemusel nic instalovat ani nastavovat. Mým cílem je, aby pouze vstoupil na webovou adresu, přičemž o nic jiného se nemusel starat. Heslo je: "Dostupnost odkudkoli a kdykoli".

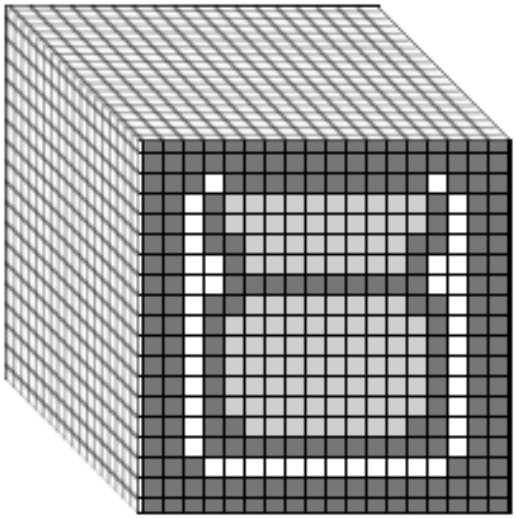
Lékař tedy přistoupí do aplikace. Ze seznamu si následně zvolí pacienta (jehož data jsou uložena na serveru) a vše se automaticky připraví. Tím odpadá i starost šíření dat mezi lékaři. Maximální komfort.

V první kapitole si rozebereme některé techniky pro zobrazení medicínských dat. Budeme se zabývat multiplanárním zobrazením a volume renderingem. V druhé kapitole se podíváme na volně dostupné programy pro prohlížení volumetrických dat. Kapitola tři přibližuje možnosti zobrazení 3D scény v okně webového prohlížeče (WebGL, Native Client). Zmiňujeme zde dostupné frameworky pro práci s WebGL, přičemž některé z nich jsou popsány podrobněji (CopperLicht, O3D). Čtvrtá kapitola popisuje návrh aplikace pro zobrazování medicínských dat. Implementace aplikace je popsána v kapitole pět.

Kapitola 2

Principy zobrazení medicínských dat v 3D

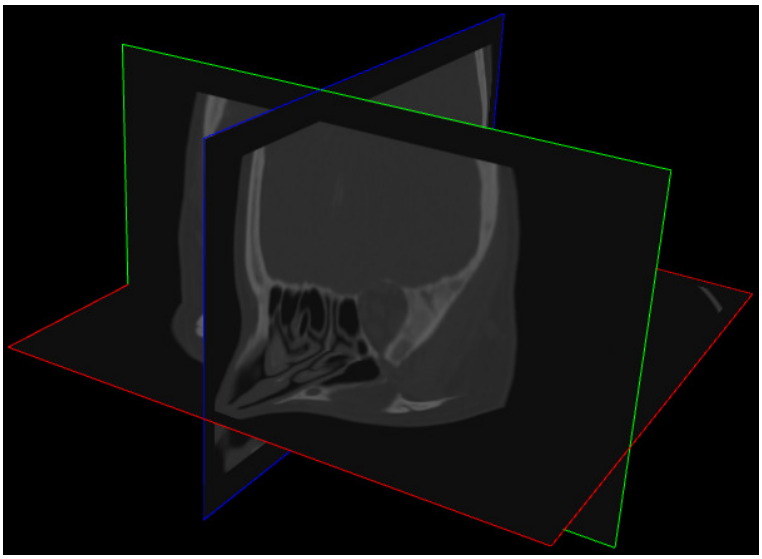
Medicínská data jsou standardně uchovávána v pravidelné ortogonální trojrozměrné mřížce. Jednotlivé voxely objemových dat pak reprezentují intenzitu odrazivosti tkáně lidského těla. V 3D grafice lze chápat tyto data jako objemovou texturu (obr. 2.1), kterou můžeme libovolně rozříznout a podívat se na data v námi specifikovaném místě.



Obrázek 2.1: Uložení medicínských dat - voxely

2.1 Multiplanární zobrazení dat

Jedním ze způsobů zobrazení medicínských dat je užití multiplanárního zobrazení. Jedná se o tři na sebe kolmé roviny, jak můžete vidět na obrázku 2.2. Jelikož jsou data uložena v ortogonální trojrozměrné mřížce (představme si jako krychli), postačí nám k bezproblémovému zobrazení celé množiny dat pouze tři na sebe kolmé roviny. Roviny označíme jako XY, XZ a YZ. Aby bylo možné zobrazit celou množinu dat, je nutná podpora pohybu s řezy. Řezy na sebe při pohybu promítají data, jenž jsou na aktuální pozici řezů v objemových datech.

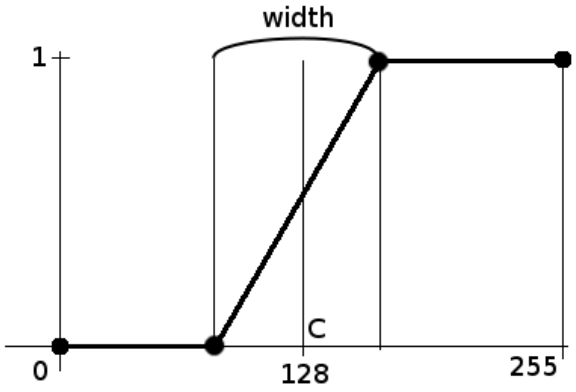


Obrázek 2.2: Multiplanární zobrazení

Je třeba uvážit zkreslení rozměrů naměřených dat, aby nedocházelo k deformaci obrazu oproti skutečnosti při jeho následné rekonstrukci.

2.2 Densitní okénko

Jedná se o mapovací funkci, která umožňuje zvýraznit měkké tkáně nebo naopak tkáně tvrdé. Její pomocí také mapujeme 8bitová data do intervalu 0–1. V O3D API se intenzita barvy vyjadřuje číslem typu double v intervalu právě 0–1 (je tomu tak zvykem i v OpenGL, pokud není vyžádáno jinak).

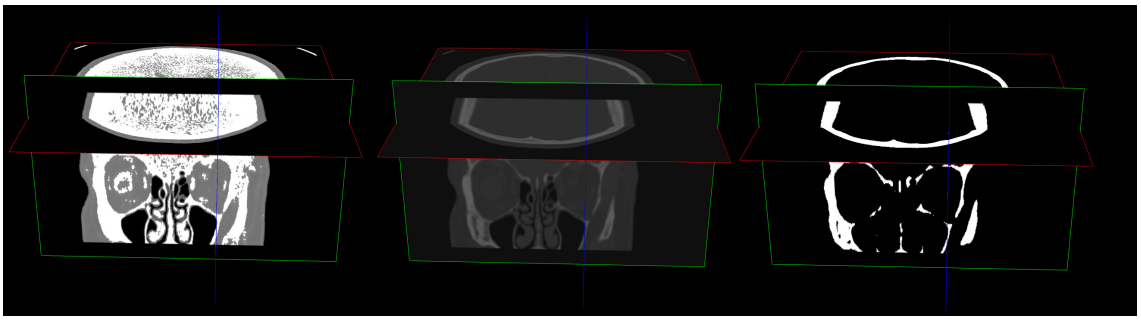


Obrázek 2.3: Mapovací funce se středem a rozsahem

V závislosti na umístění středu a rozsahu určujeme, budou-li vidět tkáně měkké nebo tvrdé. Na obrázku 2.3 je střed nastaven na střed, ale šířka není celých 8bitů. To zapříčiní větší kontrast mezi tvrdými a měkkými tkáněmi. Budeme-li mít střed umístěný uprostřed a

šířku celých 8bitů, obrázek zůstane beze změny. Posuneme-li však střed do spodní poloviny, začnou se zvýrazňovat spíše tvrdé tkáně a naopak.

Na obrázku 2.4 je názorně vidět rozdíl zobrazení při užití densitního okénka.Na obrázku vlevo jsou zvýrazněny tkáně měkké, uprostřed je původní náhled a v pravo jsou zobrazeny tkáně tvrdé. Možnost takto rozlišovat tkáně je pro lékaře velmi důležitá.



Obrázek 2.4: (zleva) : měkké tkáně, standardní zobrazení, tvrdé tkáně

Densitní okénko využívá vlastností tresholdingu(prahování). Práh je ale tvořen LUT (Look Up Table). Nejedná se tedy o tresholding doslovně. Tresholding je dán funkcí:

$$f(c) = \{ \begin{matrix} a\, pokud\, c < prah \\ b\, pokud\, c >= prah \end{matrix} \}$$

2.3 Volume rendering

Volume rendering [8] je technika používaná k reálné 2D projekci objemových dat. Výhodou volume renderingu je, že nám umožní dokonalejší chápání prostorových dat, která bychom si jinak jen obtížně představili. Jeho nevýhodou je nemožnost označení konkrétních voxelů. V tomto ohledu je výhodnější výše zmiňované multiplanární zobrazení.

Nejznámější renderovací techniky jsou: Volume ray casting, Splatting, Shear warp, Texture mapping a Hardware-accelerated volume rendering. Příklad, jak vypadají vyrenderovaná data, je na obrázku 2.5.

K renderování objemových textur se využívá podpory 3D textur na grafické kartě, avšak tato možnost není pro 3D grafiku v okně webového prohlížeče prozatím dostupná. V nejbližší době ani nebude, jelikož standard WebGL se snaží být zcela multiplatformní. Například mobilní telefony v dnešní době nemají dostatečný výpočetní výkon aby 3D textury zvládly. Pokročilé metody volume renderingu tedy nejsou přístupné. Nabízí se však možnost využít Texture mapping.

Texture mapping [12, 11] Objemová data jsou reprezentována jako balík sousedních plátů. Je-li dostupná hardwarová podpora objemových textur, lze zobrazovat pláty rovnoběžně se zobrazovací rovinou(tedy kolmé k pohledu) - obrázek 2.6. V našem případě však toto není možné. Jak bylo zmíněno již výše, podpora objemových textur ve standardu WebGL není implementována. Pláty musí být tzv. "objektově zarovnané (object-aligned slices)".

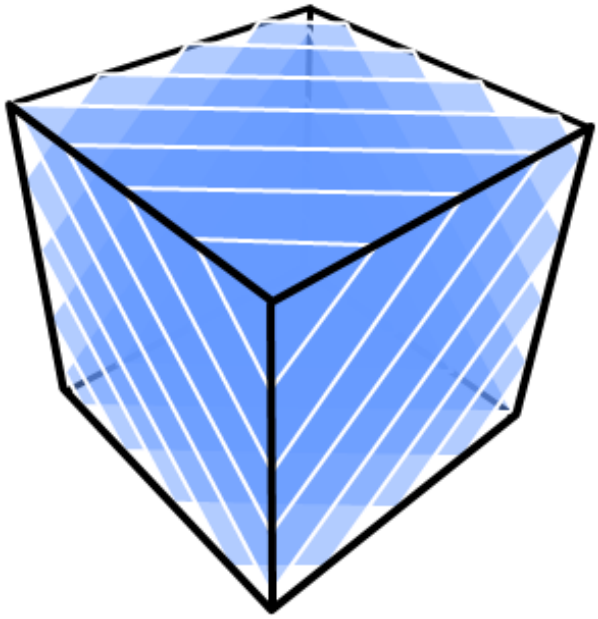
Problém texture mappingu s 2D texturami je, že pláty nemohou být vždy kolmy na směr pohledu. Zobrazení pomocí objemových textur je tedy lepší. Je-li směr pohledu změněn o více než 90 stupňů, je nutné změnit orientaci normálového vektoru. Právě proto mu-

Obrázek 2.5: Volue rendering - [15]

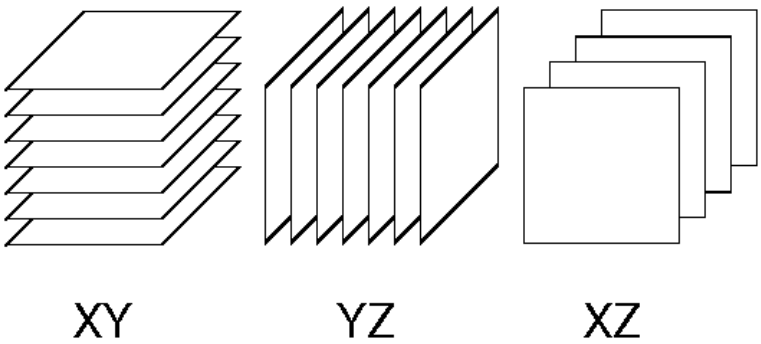
síme udržovat v paměti tři sety 2D texturových map, přičemž každý set je kolmý k jedné z hlavních os (x, y, z) - obrázek 2.7. To nám zaručuje, že v nejhorším případě budou pláty v odchylce 45 stupňů od směru pohledu.

2.4 Isosurfaces

Isosurfaces[14] je technika, která převádí volumetrická data na polygonální model. Povrch modelu je definovaný propojením voxelů o stejné intenzitě. Ta je zadána (tzv. *isovalue*). Při převodu objemových dat na polygonální model může dojít vlivem šumu v objemových datech k tvorbě nežádoucích artefaktů na výsledném modelu. Ty se musí následně vyhlazovat.



Obrázek 2.6: Pláty při užití objemových textur [16]



Obrázek 2.7: Pláty bez podpory objemových textur [16]

Kapitola 3

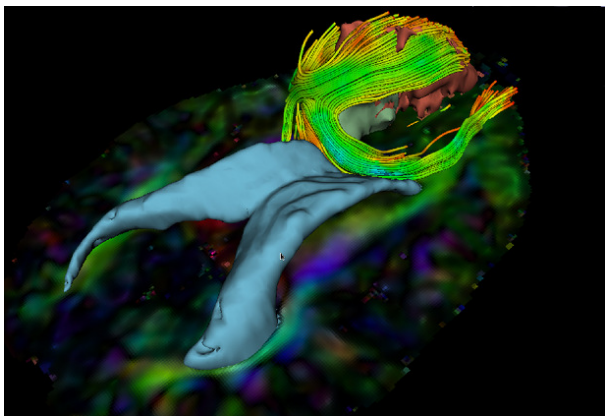
Nástroje pro vizualizaci medicínských dat

Existuje již mnoho nástrojů pro vizualizaci medicínských dat. Některé z nich zmíním níže. Uvádím je hlavně jako inspiraci pro svoji aplikaci, která ale narozdíl od nich běží online a je na konkrétním počítači zcela nezávislá.

3.1 Slicer 3D

Slicer3D[7] je program pro vizualizaci a analýzu dat. Jedná se o open-source (BSD licence) program. Je navržen multiplatformě. Bez problému běží jak na Windows, tak na Linuxu a Mac OS X.

Editor prostředí je velmi interaktivní. Samozřejmostí je podpora formátu DICOM, který je používán k uchování medicínských dat. Dostupná je i automatická segmentace dat pro snažší orientaci v datech. Podrobnější popis naleznete na domovské adrese[1] tohoto projektu. Nachází se zde velmi rozsáhlá databáze snímků programu, na kterých jsou názorně vidět jeho funkce.



Obrázek 3.1: Nové pokročilé segmentování dat (Slicer3D) [7]

[1] www.slicer.org

3.2 Drishti

Drishti[1] byl vyvinut za účelem vizualizace CT dat, elektronové-mikroskopie a dalších. Hlavním cílem projektu Drishti je, aby vědci byli schopni prozkoumávat volumetrická data (nejen lékařská). Stejně tak, aby je mohli používat ve svých prezentacích (Drishti umožňuje vytvářet i animace[2]).Podrobnější popis[3] projektu Drishty.

Obrázek 3.2: Lze zobrazovat i jiná data - mravenec [1]

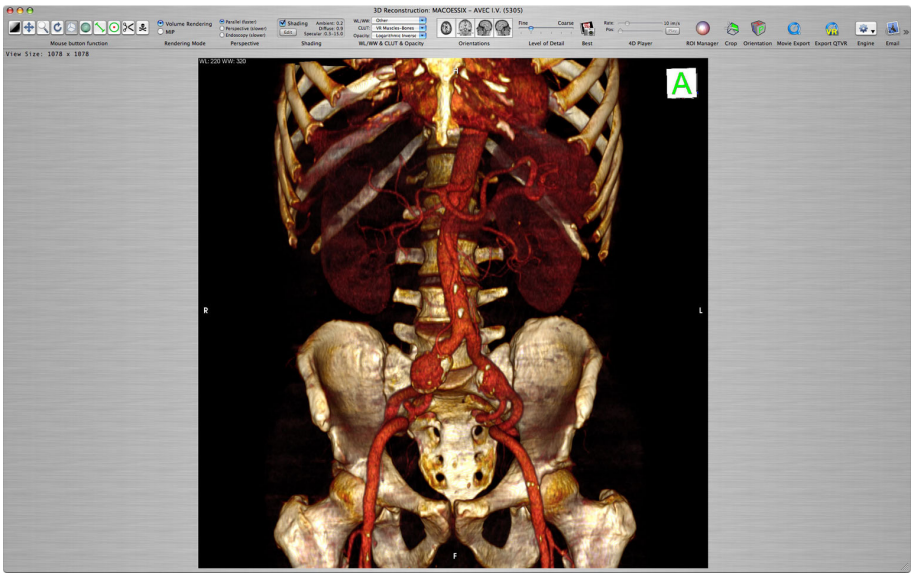
3.3 InVesalius

InVesalius[19] je software pro zobrazování medicínských dat. Je poskytován zdarma. Běží na OS Linux a Windows. Podpora pro Mac OS X bude v budoucnu také. Podporuje formát DICOM. Více informací naleznete na adrese projektu[4].

3.4 OsiriX Imaging software

OsiriX[17] je nástroj na zpracování objemových dat. Formát DICOM je podporován. Co všechno je ve formátu DICOM podporováno, naleznete zde[5]. Umožňuje zpracování i ostatních formátů (např. TIFF, JPEG, PDF, AVI, MPEG). Jeho nevýhodou je, že není multiplatformní. Je napsán pouze pro Mac OS X. Dostupný je jak v 32-bitové, tak v 64-bitové verzi. 64-bitová verze má řadu výhod. Narozdíl od 32-bitové verze umožňuje nahrát neomezený počet snímků přesahující limit 4GB. 64-bitová verze je i více optimalizována pro vícejádrové procesory, čímž znatelně zvyšuje výkon při renderování.

[2] http://anusf.anu.edu.au/Vizlab/drishti/gallery.shtml
[3] http://anusf.anu.edu.au/Vizlab/drishti/features.shtml
[4] http://svn.softwarepublico.gov.br/trac/invesalius
[5] http://www.osirix-viewer.com/AboutOsiriX.html



Obrázek 3.3: Segmentace spojená s volume renderingem (OsiriX) [17]

3.5 Seg3D

Seg3D[2] je nástroj pro segmentaci a zpracování objemových dat. Jedná se o open-source projekt, který byl založen hlavně díky "SCI Institute's NIH/NCRR CIBC Center". Seg3D kombinuje manuální nastavení segmentačního rozhraní s výkonným vícedimenzionálním zpracováním obrázků a segmentačními algoritmy. Je dostupný i volume rendering.

3.6 ParaView

ParaView[18] je open-source, který je multiplatformní. Paraview bylo vyvinuto k analýze **extrémně velkých setů dat** využívajících distrubuované paměťové zdroje. Může běžet na superpočítačích k analyzování setů dat extrémních rozlišení, stejně tak jako může běžet na běžném laptopu s malým množstvím dat.

Kapitola 4

Zobrazení 3D scény ve webovém prohlížeči

Již není třeba instalovat dodatečné plug-iny do prohlížečů pro běh akcelerované grafiky. Technologie, které to umožňují, jsou WebGL nebo Native Client.

4.1 Technologie WebGL

WebGL vychází ze standardu OpenGL ES 2.0. Původním autorem byla skupina Mozilla Foundation. V současné době se o vývoj zasluhuje skupina Khronos Group. Jedná se o multiplatformní standard. Umožňuje programování pokročilé 3D grafiky, jež běží v okně webového prohlížeče. Jedná se samozřejmě o akcelerovanou grafiku. Pro vykreslování shaderu byl zvolen standardně jazyk GLSL (OpenGL Shading Language).

Jelikož je tento standard zcela přenositelný, a to včetně mobilních zařízení (mobily, tablety), byla vynechána podpora objemových textur. To znemožňuje naprogramování pokročilého volume renderingu. Existuje však varianta texture mappingu, jak bylo vysvětleno v sekci 2.3 na straně 6, která volume rendering umožňuje.

Další nespornou výhodou WebGL je plná integrace s rozhraním DOM (Document Object Model). Znamená to, že může být využito jakýmkoliv programovacím jazykem, který podporuje rovněž DOM. Nejsme omezeni tedy pouze na Javascript, ač v současné době je to nejpoužívanější kombinace. K dispozici je rozsáhlá a velmi podrobná specifikace[1].

4.2 Frameworky pro práci s WebGL

Frameworků pro WebGL je skutečně mnoho. Za zmíňění stojí:

- SceneJS[2]
- C3DL[3]
- GLGE[4]

[1] http://www.khronos.org/registry/webgl/specs/latest/
[2] http://www.scenejs.org/
[3] http://www.c3dl.org/
[4] http://www.glge.org/

- CopperLicht[5]
- O3D[6]

Některé z nich rozeberu podrobněji.

CopperLicht

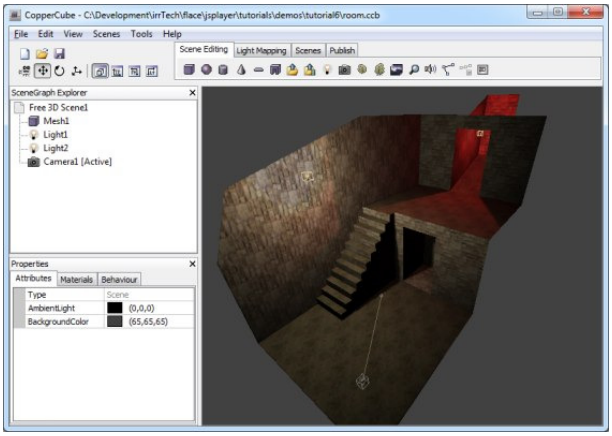
CooperLicht[5] je vyvíjen firmou Ambierra. Jedná se o komerční engine, který je pro nekomerční použití poskytován zdarma. Bohužel, bezplatně je poskytována pouze jeho komprimovaná podoba. Nekomprimovanou verzi je nutné si zakoupit.

Společnost Ambierra dodává spolu s tímto enginem i editor CopperCube, jenž výrazně ulehčuje vymodelování scény. Podle výrobce není třeba při vytváření jednodušší aplikace programovat. Ačkoli editor umí vytvářet rozsáhlé scény, jejich velikost je omezena. Je ale omezena pouze při užití tohoto editoru. Samotný engine nemá uvedena žádná omezeni, co se velikost scény týče. Další podstatnou vlastností editoru je schopnost importovat data i z jiných editorů. Podporované formáty jsou : 3ds, obj, x, lwo, b3d, csm, dae, dmf, oct, irrmesh, ms3d, my3D, mesh, lmts, bsp, md2 a stl.

Engine podporuje animátory. Tím velice ulehčuje pohyb ve scéně. Implementována je také podpora kolize objektů. Zajímavá vlastnost je tzv. "gravitace objektů", kde můžeme nadefinovat gravitační zrychlení objektu.Ten pak bude zrychlovat, neleží-li na jiném stabilním objektu. Transformace jsou velmi intuitivní. Zrovna tak mapování textur na objekty. Objekty je možno vytvořit buď standardně přes pole vertexů (ručně), nebo pomocí editoru CopperCube (ten samozřejmě taky vygeneruje vertexy, ale pro uživatele je to pohodlnější a znatelně rychlejší).

Domnívám se, že tento engine je velice vhodný pro rychlé vytvoření i rozsáhlé scény, ovšem za předpokladu, že programátor nebude mít velké nároky na kvalitu zobrazení, jelikož engine nepodporuje dynamické osvětlení (tuto vlastnost slíbil výrobce do příštích verzí dodělat).

Jak psát programy pomocí CopperLichtu naleznete na adrese projektu [7].



Obrázek 4.1: Ukázka editoru CopperCube- [5]

[5] http://www.ambiera.com/copperlicht/index.html
[6] http://code.google.com/p/o3d/
[7] http://www.ambiera.com/copperlicht/tutorials.html

O3D s WebGL

O3D[4] API je rozhraní pro tvorbu bohatých a interaktivních aplikací ve webovém prohlížeči. Celý projekt je veden pod licencí open-source. V dřívějších dobách bylo O3D distribuováno jako plugin, jenž bylo nutné doinstalovávat. Jakmile vznikl standard WebGL, bylo O3D přepsáno pro použití s touto technologií. Nutnost instalace tedy odpadá. O3D je plně kompatibilní s Javascriptem.

Scéna O3D není omezena svou velikostí. Pro umístění objektu ve scéně jsou předdefinované metody. Stejně tak pro pohyb s objekty. Rotace, posuny, scale, natáčení kamery, přibližování a mnoho dalšího je již implementováno.

Je dostupná i podpora shaderů (vertex a fragment shader). Pomocí shaderů je možné řešit i komplexnější osvětlení. Samotné shadery se v O3D musí programovat do HTML tagu <textarea>, ze kterých jsou programy načteny, zpracovány a přiřazeny konkrétnímu objektu (materiálu). Stejně tak se definuje i vertex shader.

Stejně jako v CopperLichtu je možné objekty vytvořit ručně (zapsáním vertexů do polí). Lze importovat i hotové modely. Ruční zadávání vertexů ale opravdu doporučuji jen při extrémně jednoduchých objektech(krychle, deska). Jako editor se dá použít například Google SketchUp[8].

- **Importování obsahu**[4]

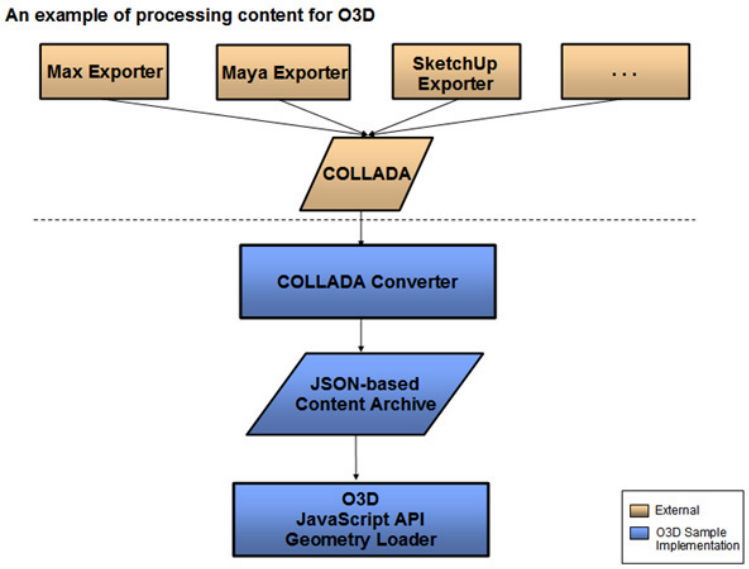
  COLLADA (COLLAaborative Design Activity)[13]
  Collada představuje formát, který umožňuje jednoduchý přenos dat mezi různými grafickými aplikacemi. Více na [13]. Je založena na formátu XML.

  COLLADA converter
  Jak je ukázáno na obrázku 4.2, "raw"collada data, která byla vyexportována programy 3ds, Maya a SketchUp, jsou konvertována "COLLADA konvertorem"pro použití v O3D API. Příklady jak importovat soubory a podrobnější popis naleznete v tomto tutoriálu[9].

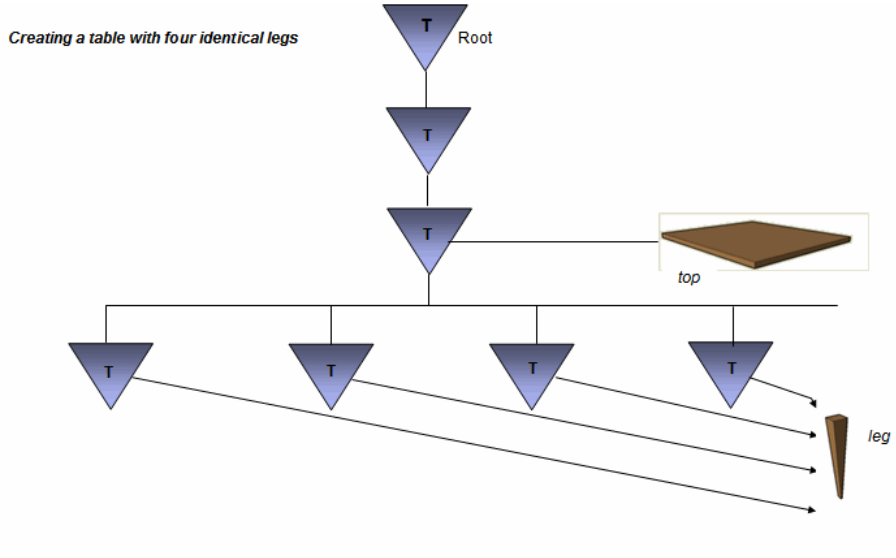
- **Scene Graph API**[4]
  O3D Scene graph API se používá k vytvoření struktur transform graph a render graph. Transform graph uchovává informace o pozici, velikosti, útvarech(objektech) ve scéně, materiálech a shaderech. Render graph uchovává informace o tom, jak 3D objekty konvertovat (renderovat) do "pixelů", jenž jsou následně zobrazeny na obrazovce.

  Jak se vytváří objekty[4]
  Uveďmě si to na příkladu (obr. 4.3). Vytvoříme shape (pomocí zobrazovacích seznamů, přiřadíme efekty, materiál atd.). Přidáme ho do transformace (transform graph). Daný útvar můžeme přiřadit i jiným transformacím (na našem obrázku je tímto útvarem noha u stolu). Tyto 4 transformace(nohy) přiřadíme pod další transformaci, do které přiřadíme další útvar (desku stolu). Nakonec vytvoříme finální transformaci, která bude zastupovat celý stůl. Výhodou tototo přístupu je následná flexibilita objektu. Stačí když manipulujeme s nejvýše nadřazenou transformací a tím manipulujeme se všemi objekty v transformaci obsažené (hýbeme s celým stolem). Hlavní uzel transform graphu se nazývá root. Na ten se připojují všechny ostatní transformace ve scéně.

[8] http://sketchup.google.com/
[9] http://code.google.com/intl/cs-CZ/apis/o3d/docs/artdesignerguide.html



Obrázek 4.2: Ukázka zpracování obsahu pro O3D [4]



Obrázek 4.3: Ilustrace vytvoření objektu [4]

Podrobnější informace o O3D API naleznete na této adrese[10].

[10] http://code.google.com/intl/cs-CZ/apis/o3d/docs/techoverview.html

4.3 Podpora WebGL v prohlížečích

Podpora v prohlížečích není příliš rozšířená. Postupně se však začíná prosazovat.

Firefox 4.0

V nejnovější verzi podporuje standard WebGL i Firefox. Jedná se o verzi 4.0. Je však stále nutné podporu explicitně zapnout. To se provádí tak, že do příkazového řádku zapíšeme "`about:config`". Pak vyhledáme položku webgl a změníme hodnotu položky "webgl.enabled_for_all_sites"na `true`.

Google Chrome

Google Chrome podporuje WebGL experimentálně již od verze 8. Bylo jej však nutné zapínat s parametrem "–enable-webgl". Ve verzi 9 je nutné používat i parametr "–ignore-gpu-blacklist". Na OS Windows by od verze 10 již neměli být žádné problémy a aplikace by měla běžet zcela bez obtíží. Na OS s jádrem Linux je stále nutné používat výše zmíněné parametry.

Google Chrome je pro pokročilé programy pravděpodobně tou nejlepší volbou. V dnešní době má nejrychlejší interpretaci Javascriptu.

4.4 Native Client SDK

Native Client[3] nesouvisí nijak s technologií WebGL. Umožňuje bezpečné provádění platformě nezávislých programů v prostředí webového prohlížeče. Bezpečnost je zaručena použitím "sandboxing"technologie, při níž dochází ke kompletnímu odstínění spouštěného softwaru od zbytku systému. Jedná se o jistý styl "virtualizace".

Native Client umožňuje spouštění výpočetně náročných aplikací jako jsou hry, práce s médii, rozsáhlá analýza dat a další, protože nepracuje s javascriptem, ale s nativním (mnohem rychlejším) kódem - například jazyk C.

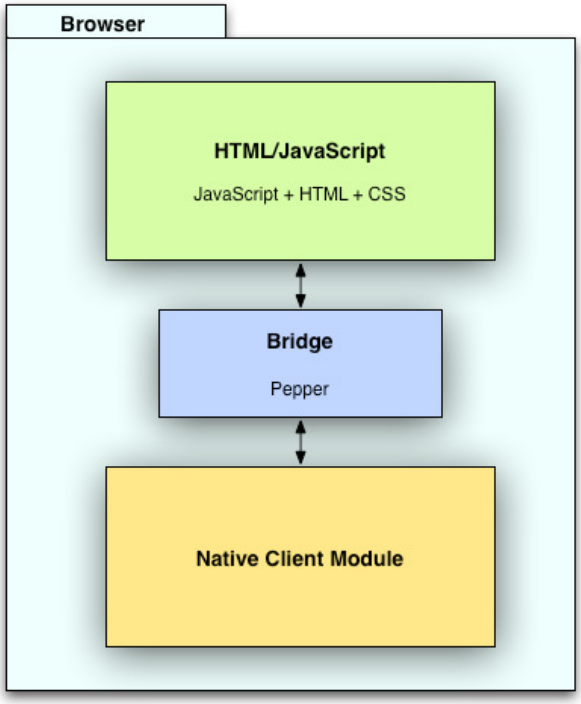
Umožňuje i práci přímo s OpenGL.

Architektura aplikace

- **HTML/JavaScript** - zajišťuje uživatelské rozhraní a uživatelské vstupy
- **Most(Pepper)** - zajišťuje komunikaci mezi Javascriptem a modulem Native Clienta
- **Modul Native Clienta** - zajišťuje časově náročné výpočty (běží zde nativní kód aplikace)

Podrobnější popis je dostupný na stránce projektu[11]. Příklady[12] včetně návodu jak je zprovoznit.

[11] http://code.google.com/intl/cs-CZ/chrome/nativeclient/docs/technical_overview.html
[12] http://code.google.com/intl/cs-CZ/chrome/nativeclient/docs/examples.html



Obrázek 4.4: Architektura aplikace při užití Native Client SDK[3]

Kapitola 5

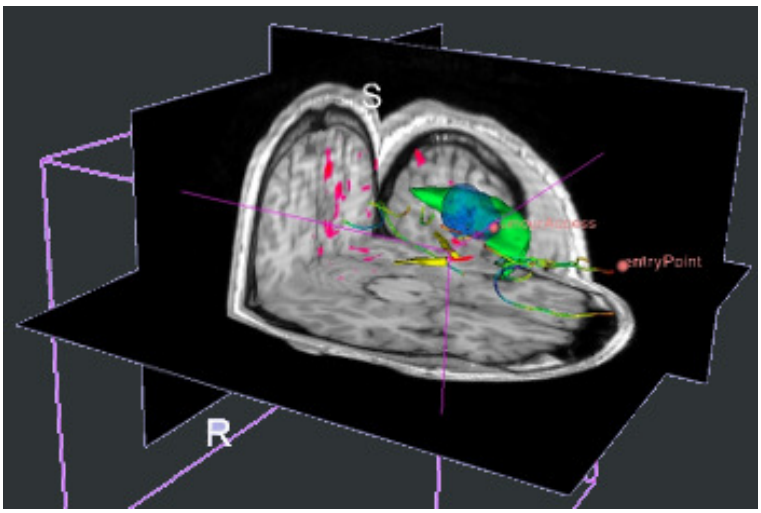
Návrh aplikace pro zobrazení medicínských dat

Jak bylo psáno úvodem, za cíl své bakalářské práce jsem si zvolil tvorbu aplikace pro multiplanární zobrazení medicínských dat.

5.1 Vize aplikace

Základní představu jak by měla aplikace vypadat jsem získal inspirací z programů již hotových. Jako příklad jsem v sekci 3 uvedl programy Slicer3D, Drishti a InVesalius. Tyto programy jsou samozřejmě pokročilejší než mnou plánovaná aplikace. Podporují například pokročilý volume rendering. Má aplikace je narozdíl od nich mnohem přívětivější k uživateli. Mým cílem bylo, aby uživatel pouze přistoupil na adresu aplikace a ihned mohl pracovat s medicínskými daty. Odpadá tak i starost, jak šířit data mezi lékaři.

Jak by měla aplikace zhruba vypadat po dokončení vidíte na obrázku 5.1. Obrázek je z programu slicer3D (zde je ale vidět i segmentace, kterou do svého programu aktuálně zahrnovat neplánuji).



Obrázek 5.1: Vize aplikace

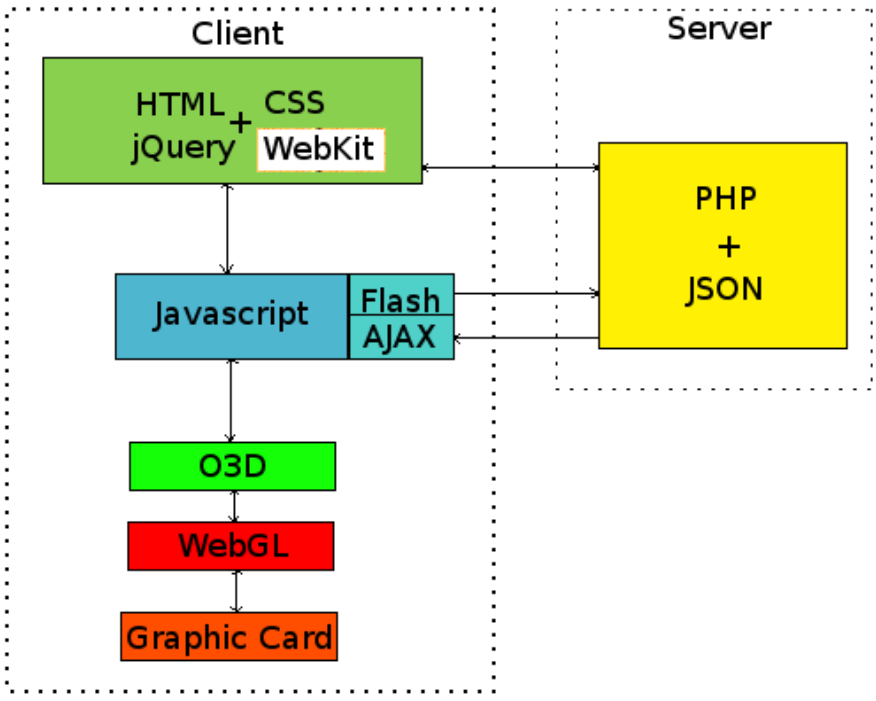
5.2 Volba technologií

Vzhledem k cíli aplikace, která zahrnuje i maximální komfort uživatele, okamžitě odpadla možnost využití některého z pluginů (pro pokročilou grafiku), jež se musí doinstalovávat. Existují tedy dvě možnosti. Použití technologie Native Client a psaní aplikace v nativním kódu nebo použití nového standardu WebGL.

Standard WebGL je lepší volbou, jelikož Native Client je opravdu v ranné fázi vývoje. Pro programování 3D grafiky jsme si zvolili tedy WebGL. Nesmíme ale zapomenout zmínit i další zcela nezbytné technologie. Naprosto základní je HTML, které nám umožňuje udržovat rozvržení aplikace. K přesnějšímu rozvržení a hezčímu vzhledu je použita technologie CSS. Jelikož by programování v čistém WebGL bylo náročné a zdlouhavé, používáme API O3D, který nám práci velmi ulehčuje. Celá aplikace je napsána v jazyku Javascript, který zprostředkovává komunikaci mezi DOM a O3D.

Pro nahrávání dat byla použita knihovna, která využívá technologii Flash. A to z toho důvodu, aby mohl uživatel vybrat více souborů naráz.
Na straně serveru běží PHP skript, který zprostředkovává jak ukládání nových dat, tak zasílání dat ze serveru na požádání klientovi.
Naprosto nezbytnou technologií se ukázal AJAX, který umožňuje provádět mnohé věci bez obnovení stránky (k tomu patří i získávání informací o datech ze serveru).

Aplikaci využívá i framework jQuery, který usnadňuje práci s javascriptem a poskytuje rozšířené rozhraní. Pro uchovávání informací o datech používá aplikace JSON.



Obrázek 5.2: Schéma propojení použitých technologií

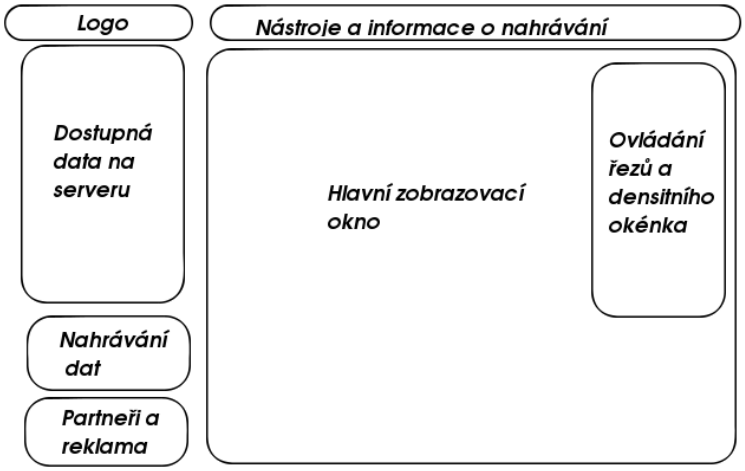
5.3 Rozvržení aplikace

Aplikace byla navržena co nejjednodušeji, avšak s ohledem na to, aby zůstala zcela plnohodnotnou. Přibližné rozvržení je na obrázku 5.3

Největší prostor je vymezen zobrazovacímu canvasu. Ten je částečně překrýván ovládacími prvky. Jedná se o posuvníky pro manipulaci s řezy, posuvníky pro ovládání densitního okénka a tlačítka na resetování objektu do původního stavu.

V horní části je umístěn tenký proužek. V jeho levé části je volný prostor pro umístění loga aplikace. Větší pravá část bude použita pro umístění dalších ovládacích prvků (zoom, nastavení cache) a průběhu, který udává informaci o stavu načítání dat ze serveru.

Zcela nalevo jsou dialogy potřebné ke zvolení dat, která chceme prohlížet (včetně podrobných informací o datech). Dále dialog pro nahrávání nových dat na server. V neposlední řadě je třeba vymezit jistou část aplikace pro prezentaci partnerů a reklamy(v běžných aplikacích to není třeba, ale u webových ano).



Obrázek 5.3: Návrh rozvržení prvků aplikace

5.4 Ovládání aplikace

Intuitivnost ovládání může rozhodovat o uplatnění aplikace na trhu, ať je její zbytek jakkoli dobrý. Proto mu byla věnována zvláštní pozornost. Samozřejmostí jsou posuvníky pro posun s řezy a nastavování prahu (densitního okénka). Nebylo ani opomenuto velmi jemné ovládání(posunování o jeden řez). Existuje však další varianta jak program ovládat. Je to velmi rychlé a příjemné.

Preferovaný způsob ovládání je pomocí myši a stisknuté klávesy.
Rotaci objektu provedeme pouhým stisknutím myši nad ním a jejím tahem. Chceme-li posunovat s jednotlivými řezy, přidržíme klávesu levý control, stiskneme tlačítko myši nad daným řezem a táhneme požadovaným směrem (nutno dodat že se jedná o experimentální funkci). Přibližování provádíme pomocí kolečka myši (na noteboku lze použit obvykle pravou část touchpadu). Je také implementován pohyb s kamerou. Přidržením klávesy levý shift, stiknutým tlačítkem myši a následným tahem natáčíme kameru (rozhlížíme se). Toto je velmi důležitá vlastnost, abychom mohli zobrazit celou množinu dat i při velkém přiblížení objektu.

5.5 Vlastnosti aplikace:

- **načítání dat ze serveru**: Veškerá data o pacientech budou uložena na centrálním serveru. Lékař zvolí požadovaná data a ty se mu nahrají do počítače.
- **multiplanární zobrazení medicínských dat**: Způsob zobrazení volumetrických dat. Podrobnější popis je v sekci 2.1 na straně 4.
- **volume rendering**: Prostorové renderování volumetrických dat. Podrobnější popis je v sekci 2.3 na straně 6.
- **isosurfaces**: Převod volumetrických dat na polygonální model. Více v sekci 2.4.
- **využití serveru pro složité výpočty**: Zvážení možnosti využití serveru pro náročné výpočty (segmentace, renderování, vytvoření modelu). Jistá alternativa by se mohla skrývat v Native Clientu.
- **správa účtů**: Správou účtů by se docílilo větší personalizace dat a rozšířilo se použití programu. Samozřejmě vše s odpovídajícím zabezpečením.
- **konferenční spojení**[6]: Zajímavým doplňkem by bezpochyby bylo umožnění spolupráce skupině uživatelů. Konference by probíhala zhruba následovně. Byl by jeden prezentující, jenž by mohl přímo do dat zakreslovat a zapisovat poznámky. Ostatní by real-time viděli práci prezentujícího a do diskuse by mohli přidávat příspěvky. Stálo by za uvážení implementovat možnost "předání"pera jinému účastníkovi prezentace, aby i on mohl názorně zakreslit své nápady.
- **automatická segmentace dat**: Automatická segmentace dat s následným označením od uživatele je také praktická. V tomto případě by mohl program sloužit i jako naučná pomůcka. Studenti by mohli studovat reálná data okomentovaná pedagogem.

Plánované vlastnosti v rámci BP

V rámci bakalářské práce vznikne základ programu, který bude obsahovat následující vlastnosti z výše jmenovaných:

- načítání dat ze serveru
- multiplanární zobrazení medicínských dat

Jedná se o klíčové vlastnosti, které umožňují zobrazení celé množiny libovolných volumetrických dat.

Kapitola 6

Implementace aplikace

Aplikace je implementována téměř celá objektově. Základní věci jako inicializaci aplikace, správu paměti, získání a dekompresi dat, generování textur a jejich následnou projekci i implementaci ovládání popíši níže.

6.1 Inicializace aplikace

Celé API O3D je implementováno v knihovně s názvem o3djs. Tu vkládáme ihned při načtení stránky. Následně si z ní nahrajeme potřebné komponenty. Nás se týkají "*util, webgl, rendergraph, primitives, material, picking, arcball, element a debug*". Další moduly které je nutno vložit jsou: "*mouseaction, keyaction, createobject, getdata, helpFunction, projectionTools, textureHandler, objectManipulation.* Podrobnější popis těchto modulů je uveden níže.

Po inicializaci prvotních nastavení, spustíme metodu *makeClients*, ta zkontroluje zda-li je v počítači dostupná dostatečná podpora pro správný běh aplikace.Po kontrole volá funkci *main. Main* řídí inicializaci celé aplikace. Nejdříve si zjistí reference na posuvníky(ovládající polohu řezů). Reference pomocí metody *setSlider* objektu *manipulation* uloží. Následně uloží do bufferů aktuální hodnoty posuvníků pomocí metody *setLast.* Následuje inicializace globálních proměnných ve funkci *initGlobalVar.* V této funkci dochází také k prvotnímu nastavení kamery a arcballu. Pokračujme dále ve funkci main. Přepneme styl vekreslování na "RENDERMODE_ON_DEMAND". Tím docílíme rapidního snížení zátěže na systém, protože budeme vykreslovat pouze na požádání. Nyní přichází na řadu metoda *createObjects* objektu *CreateObjects.* Následně nastavíme textury na vzniklé útvary a necháme objekty znovu překreslit. Musíme také inicializovat *pickManager* a přizpůsobit ho současné scéně pomocí metody *update*, která je součástí přímo knihovny O3D. Zapamatujeme si počáteční matici modelview a inicializace je hotová. Ještě nesmíme zapomenout na nastavení handlerů pro obsluhu uživatelského vstupu.

Objekty

CreateObject

Vytváří objekt složený ze tří na sebe kolmých průměten (multiplanární zobrazení). Obsahuje veřejnou metodu *createShapes*, která má tři(dx, dy, dz) parametry, jimiž se určuje „scale“ objemové textury. Jednotlivé průmětny automaticky přidává do globálního stromu transformací uchovávájícího informace o objektech ve scéně.

GetData Objekt pro stažení dat ze serveru. Zároveň obsahuje metody pro dekompresi obrázku a vytažení matice s hodnotami jednotlivých pixelů. Umožňuje nastavovat, zda-li se bude používat cache. K tomu slouží metoda *turnCacheOn*, *turnCacheOff* a pomocí metody *statusCache* zjišťujeme, je-li cache povolena nebo ne. Informace o dostupných datech získáme metodou *showDataStructure*. Celé nahrávání dat programátor inicializuje metodou *startReadingFiles*. Jejími vstupními parametry jsou struktura obsahující informace o dostupných datech a id dat, která chceme nahrát. Jakmile jsou všechna data načtena, objekt volá globální funkci *init*. Načtená data jsou dostupná metodou *getCube*.

KeyboardHandler
Objekt obsluhující vstupy klávesnice. Obsahuje veřejné metody *keyDownAnalyze* a *keyUpAnalyze*.

MouseHandler
Objekt obsluhující události myši. Disponuje třemi veřejnými metodami. *startDrag, drag a stopDrag*. V závislosti na indikaci stisknutých kláves provádíme rotaci, posun s řezy nebo natáčení kamerou. Kolečkem myši nastavujeme úroveň přiblížení.

ObjectsManipulation
Objekt zaštiťující rotaci s objektem a pohybem s jeho řezy.

ProjectionTools
Objekt sloužící k projekci bodů z lokálního prostoru (3D bodu) do roviny obrazovky. Dostupné jsou metody *projectPoint* a *collapseVector*.

TextureHandler
Stará se o generování 2D řezů z objemových dat a jejich správné nanášení na jednotlivé řezy. Nachází se zde i metoda pro vytvoření LUT (Look Up Table) a vyhledávání případné hodnoty v ní. Dostupné metody jsou *refreshTextures, setTextureTypes, setTexture, createMap, getMapValue*.

6.2 Management paměti

Javascript využívá garbage collector. Tím odpadá mnoho kritických chyb a rozsáhlé úniky paměti. Klíčové je, aby garbage collector správně rozpoznal, která část paměti již není potřeba. To se může stát i kamenem úrazu. S tímto problémem jsem se potýkal delší dobu. Je třeba označit proměnnou jako neplatnou (nesmí na ní existovat žádná reference). Obvzláště u proměnných, které zabírají kriticky velké množství paměti (objemové textury).

Paměťová náročnost programu je poměrně velká. To je ale pochopitelné, jelikož musíme v paměti udržovat informaci o barvě jednotlivých voxelů. Z počátku jsem kopíroval a upravoval obrazové matice do trojrozměrného pole. To bylo ale naprosto neúnosné, jelikož garbage collector uvolňoval paměť příliš pozdě. Dříve než ji uvolnil, došlo k zahlcení paměti a pádu(přinejlepším) aplikace.

Jedním z řešení je ukládat do pole pouze reference na již existující matice. Při troše představivosti opět máme objemovou kostku, aniž bychom museli kopírovat data do trojrozměrného pole. Odpadá nám tak i starost s uvolňováním paměti. Velikost ostatních částí je zanedbatelná, a proto jí můžeme zcela ponechat garbage collectoru.

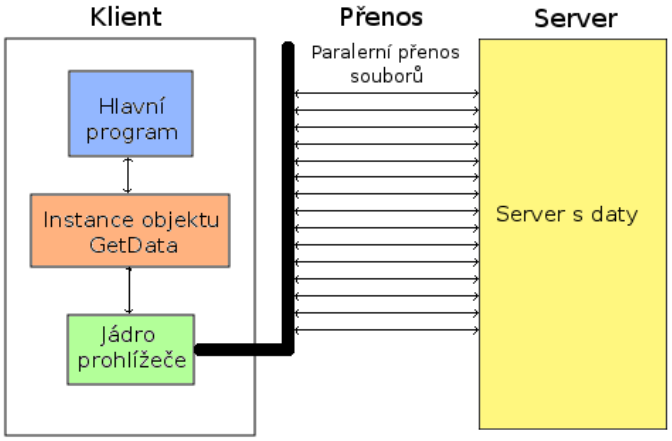
Způsob jak označit proměnnou za neplatnou:
Máme proměnnou A. Pak A = null; označí proměnnou jako nepotřebnou. Je důležité odstranit všechny reference na paměť. Tedy pokud jsme někde napsali B = A, pak musíme napsat i B = null. Jinak paměť nebude uvolněna.

6.3 Dekomprese dat

Data jsou uložena ve skupinách. Každá skupina obsahuje 2D textury (1 textura = 1řez). Formát dat je omezen pouze podporou formátů webového prohlížeče. Zaručená podpora je u obrázků PNG. Vždy se v aplikaci kódují jako RGBA, takže barevné obrázky nejsou problém.

Velkým problémem bylo, jak přenést data ze serveru ke klientovi. Dekomprese dat na serveru je poměrně rychlá. Posílat však takto velký objem dat je velmi nepraktické. Otázkou pak také bylo, v jakém formátu dekomprimovaná data přenést. Všeobecně se pro přenos dat pomocí technologie AJAX používá standard XML a JSON. Kódovat ale binární data do textové podoby (kde tak rapidně naroste jejich velikost) je nepřípustné. V úvahu by přicházela ZIP komprese na serveru a následná dekomprese na klientovi. Přenos dat by tak rapidně klesl. Problém ale nastává v rychlosti dekomprese dat pomocí Javascriptu. Po pokusech s touto technologií od ní bylo ustoupeno. Poslední běžnou možností je přenos pomocí kódování Base64 (binární data zakódovaná pomocí ascii) - nárust velikosti cca 30%. Pro nás je to stále příliš.

Za optimální řešení považuji požádat jádro prohlížeče, aby stáhlo požadovaný soubor ze serveru. Prohlížeč tak stáhne soubor PNG (velmi malý provoz na síti). Následně vytvoříme imaginární prostor pro obrázek a "virtuálně obrázek vykreslíme". Vše je velmi rychlé. Jakmile dojde k vykreslení obrázku, požádáme prohlížeč aby nám předal ukazatel na obrazovou matici(automaticky to převede do pole přístupné z Javascriptu). Výhodou použití prohlížeče pro stahování a dekompresi dat je také paralerismus.

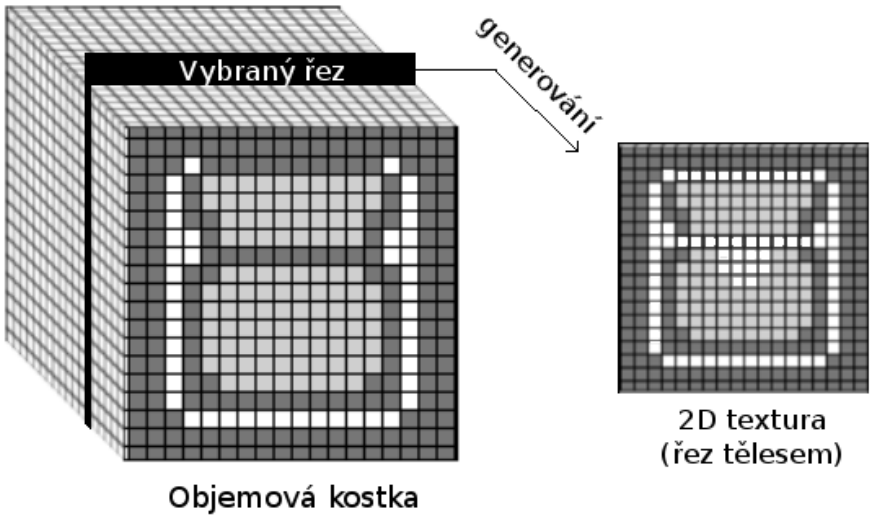


Obrázek 6.1: Přenos dat

6.4 Generování dat

2D textury, které následně vykreslujeme na plochy řezů získáváme z objemové kostky. Reprezentace objemové kostky v paměti ale není trojrozměrná. Matice je reprezentována jako souvislé pole. Horizontální řez lze získat jednoduše. Číslo řezu, který chceme získat, se rovná indexu reference v poli, ve kterém jednotlivé reference odkazují na obrazové matice(řezy). S vertikálními řezy je to složitější. Řez, ze kterého se bude aktuálně číst, získáváme jako u horizontálního řezu. Aktuální pixel v daném řezu určujeme dle offsetu. Offset počítáme dle následujících vzorců.

- Rovina YZ:
  `offset = poziceRezu * (vyskaObjektu * 4) + y * 4`
  Číslem 4 násobíme protože se pohybujeme s RGBA.
- Rovina XZ:
  `offset = x *(sirkaObjektu * 4) + poziceRezu * 4`
  Číslem 4 násobíme protože se pohybujeme s RGBA.



Obrázek 6.2: Generování textury

6.5 Projekce dat

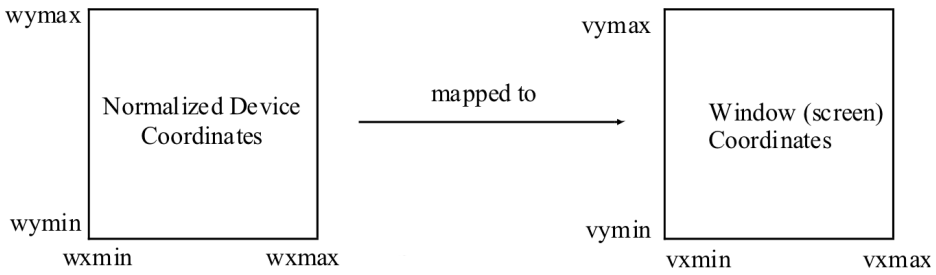
V některých případech potřebujeme sklopit bod z 3D prostoru do roviny obrazovky. K tomu nám slouží objekt ProjectionTools.

Bod sklápíme následovně:

- m = zjistíme si modelViewProjection matici
- $novy$ = vynásobíme bod v prostoru maticí m

- Provedeme výpočet souřadnic x a y: [9]

$$vx = ((vx_{max} - vx_{min})/(wx_{max} - wx_{min})) * (wx - wx_{min}) + vx_{min}$$
$$vy = ((vy_{max} - vy_{min})/(wy_{max} - wy_{min})) * (wy - wy_{min}) + vy_{min}$$



Obrázek 6.3: Mapování souřadnic[9]

6.6 Implementace ovládání

Signály od uživatele odchytáváme, předáme k další analýze a následně provedeme požadovanou akci.

Obsluha klávesnice
Signály od uživatele odchytáváme pomocí knihovny O3D. Jedná se speciálně o modul `event`. Příkaz pro odchycení události:

- `o3djs.event.addEventListener(global03dElement, 'keydown', keyboard.keyDownAnalyze);`
- `o3djs.event.addEventListener(global03dElement, 'keyup', keyboard.keyUpAnalyze);`

Při stisku klávesy je volána metoda keyDownAnalyze a při jejím uvolnění je volána metoda keyUpAnalyze. Ovládání klávesnicí slouží pouze pro nastavení indikátorů (kombinace control, shift a myši).

Obsluha myši
Signály odchytáváme stejně jako u klávesnice. Metody, které jsou volány, byly zmíněny již v sekci 6.1. Nyní si tyto metody podrobněji rozebereme.

- **startDrag**: je inicializační funkcí pro následné operace s myší. Ze všecho nejdříve pomocí pickManageru (sekce 6.1, knihovna picking v O3D) zjistíme, kam přesně bylo kliknuto v 3D. Pokud bylo myší kliknuto na některý z objektů, pak spočítáme normálu trojúhelníku (jedna promítací deska = 2 trojúhelníky), na něž bylo kliknuto. Normálu normalizujeme, pamatujeme a spočítáme také její sklopenou variantu do roviny obrazovky (jak sklápíme bod do roviny obrazovky viz. 6.5). V metodě startDrag také indikujeme, zda-li je současně stisknuta některá z kláves (control, shift) a podle toho nastavujeme příznaky pro další zpracování.

- **drag**: V této funkci nastávají tři situace. Jedná-li se o rotaci, je volána metoda *rotate* objektu Manipulations. Při aktivní detekci klávesy control, je sklopena aktuální normála (délka normály v rovině obrazovky se mění v závislosti na přiblížení desky od kamery) a poté volána metoda *moveCut*. Chce-li uživatel pohybovat s kamerou, je volána metoda *moveClient*. Zapamatujeme si aktuální pozici myši, kterou využíváme následující výpočet, a na závěr překreslíme scénu.

- **stopDrag**
  Pouze nastavuje řídící proměnné na false. Nastavuje jako defaultní rotaci objektu.

Pomocné metody:

- **rotace**
  Knihovna O3D obsahuje *arcball*. Lze si ho představit jako kulovou plochu, překrývající náš objekt. Ze všeho nejdříve musíme arcball vytvořit.

  Příkaz pro vytvoření arcballu:
  `globalBall = o3djs.arcball.create(542, 542);`

  Jeho rozměry můžeme později pozměnit:
  `globalBall.setAreaSize(globalClient.width, globalClient.height);`

  Jakmile dojde ke stisknutí tlačítka myši, je do arcballu uložena pozice kliknutí. Následně při tahu myši, je pomocí metody `globalBall.drag([event.x,event.y])` získán směr, kterým se myš pohybuje. Získáme tzv. *quaternion*. Ty převedeme metodou *quaternionToRotation* objektu *qaternions* na matici, určující směr rotace. Pak jen vynásobíme modelview matici touto maticí a získáme nové natočení objektu.

- **moveCut**
  Metoda *moveCut* zjišťuje směr a vzdálenost, o kterou je třeba řez posunout. Jejími vstupnímy parametry jsou: identifikace objektu, nové souřadnice myši, minulé souřadnice myši a aktuální sklopená normála v rovině obrazovky, která má v lokálním 3D prostoru délku 1 (posun o 1 řez). Nejdříve spočítáme vektor, jenž nám říká rozdíl mezi minulou pozicí myši a aktuální pozicí. Následně spočítáme cosinus úhlu mezi vektorem posunu myši a sklopenou normálou (normála ukazuje stále stejným směrem, takže by se měl řez posunovat správným směrem, ať je objekt natočen jakkoli). Je také třeba spočítat, o kolik řezů bude třeba posunout. Narozdíl od posunu objektu, který je určován v `double`, musíme informaci o počtu posunutých řezů uchovávat celočíselně.

  Počet řezů se tedy rovná:
  `vectAct` - vektor posunu myši
  `colNormal` - vektor normály sklopené do roviny obrazovky
  `pocetRezu = (globalMath.length(vectAct) / globalMath.length(colNormal));`

  Provedeme další korekce posunu, například v závislosti na poměru os voxelů, a pohneme s objektem.

Upozornění: Pohyb řezů pomocí myši je zatím experimentální. Správně funguje ve výchozí pozici objektu.

- **moveClient**
  Metoda *moveClient* umožňuje uživateli pohybovat s kamerou, pomocí níž sleduje scénu. Je to neocenitelná funkce obvzláště, chce-li si prohlédnout detailní povrch tělesa. Je třeba hlavně v případě, že se uživatel přiblíží k objektu natolik, že není možno zobrazovat celou množinu bodů na ploše obrazovky. Pak se uživatel "rozhlíží". Implementace je velmi jednoduchá. Pouze zjišťujeme směr myši a následně měníme cílový bod (také označován jako `target`). Jakmile nastavíme potřebné parametry, musíme updatovat modelview matici a překreslit scénu.

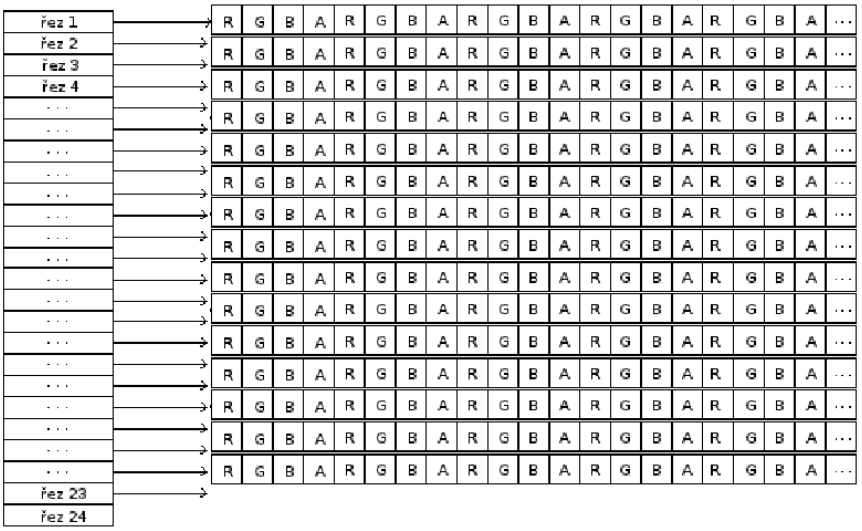
6.7 Optimalizace

V počátcích programu byla paměťová náročnost enormní. Objemová kostka s černobílými kanály barev zabírala více než 1GB RAM. Po delším pátrání byl odhalen problém s garbage collectorem. Ten uvolňoval paměť až příliš pozdě, čímž došlo k velmi rychlému zahlcení. Paměťová náročnost byla také velká z důvodu neoptimálního způsobu přenosu dat, kdy byla birnární data přenášena ze serveru přes AJAX v JSONu. I od tohoto přenosu dat bylo upuštěno. Současné stahování dat je popsáno v sekci 6.3. Pravděpodobně se jedná o jeden z nejúspornějších způsobů. Pro další optimalizace by se muselo přistoupit k jinému kódování dat. Například zavést podporu pro soubory DICOM, které by se bez problému přes AJAX nebo novou technologii websockets přenesly.

Jakmile se podařilo přenést data s dostatečnou rychlostí a přijatelnou pamětovou náročností, objevil se problém s generováním dat. To bylo extrémně pomalé. Optimalizace spočívala v zjednodušení procházení matic. Matice jsme převedli na jednorozměrná pole. Celá objemová kostka je ve skutečnosti pouze dvourozměrné pole. To je názorně vidět na obr. 6.4. Obrázek je pouze ilustrační. Vlevo je vidět jedno pole s referencemi. Vpravo je znázorněno pole, jenž obsahuje informace o jednom řezu. Jelikož je práce s objekty náročná, bylo upuštěno od zapouzdření jednotlivých pixelu do struktur(objektů) a jednotlivé složky barev jsou skládány za sebe po čtveřicích. V současné době se ještě nabízí možnost vypuštění alpha kanálu. Čímž by teoreticky došlo ke zrychlení výpočtu o 25%. Z důvodu budoucího vývoje ale byl tento kanál ponechán. Je možné, že vzniknou dva oddělené moduly. Jeden pro klasické vykreslování s vypuštěným alpha kanálem a druhý pro texture mapping, kde je průhlednost zapotřebí. To je ale ještě budoucnost. I po těchto optimalizacích byl však program velmi pomalý. Největší ztráta byla právě při generování nových textur. Zbytek programu běžel dostatečně svižně. Následující krok velmi znepřehlednil kód, ale výrazně zrychlil chod celého programu. Proměnné se museli stát globálními, jelikož dynamická alokace je velmi náročná. Nejdříve bylo přistoupeno k alokování polí přesných velikostí. Již to zrychlilo program. Jakmile se ale udělaly pole globální, čímž jsme obešli alokaci nových polí pro každou texturu, došlo k zrychlení o 300%. Opětovná alokace polí byl velký nedostatek.

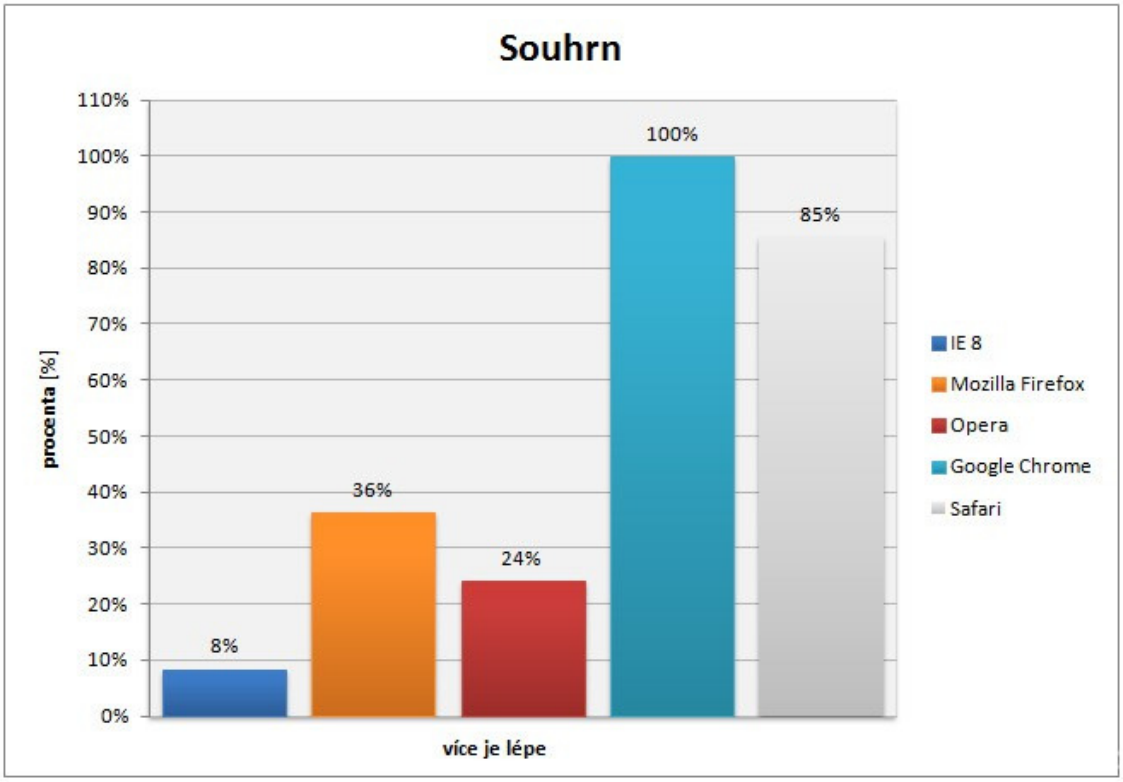
Jednou z posledních optimalizací, bylo přepsání větší části programu do objektového programování. Z počátku byl program psán strukturovaně. To bylo ale neudržitelné, jak se program rozrůstal. Nyní zůstala strukturovaně jen řídící funkce programu, která využívá dostupné objekty.

Za velkou nevýhodu považují přílišné provázání objektů globálními proměnnými. Ty jsou však prozatím naprosto nezbytné, jelikož i z objektů je nutný přímý přístup do O3D API. Tato "svázanost"je cena za nutnou rychlost.



Obrázek 6.4: Uložení dat v paměti - pouze orientační

Pevně věřím v další zrychlování Javascriptu. Srovnání rychlosti zpracování je vidět na obrázku 6.5. Jak je vidět v současné době vyniká Google Chrome společně se Safari (oba používají V8). I přesto, že program funguje ve Firefoxu, je zde naprosto nepoužitelný z důvodu rychlosti generování dat.



Obrázek 6.5: Srovnání výkonu webových prohlížečů [10]

Kapitola 7

Výsledky

Výsledkem mé bakalářské práce je program pro zobrazování medicínských dat dostupný na adrese:

www.showmedicaldata.eu

Jak program vypadá, je vidět na obrázku 7.1. Byl co nejvíce zachován základní návrh programu. I přes drobné zesložitění programu zůstalo ovládání stále velmi jednoduché. Na obrázku je konkrétně zobrazena pánev s nastaveným prahováním densitního okénka pro snadnější rozpoznání tvrdých tkání. Ovládání je podrobně popsáno v sekci 5.4.

Bylo zjištěno, že je potřeba snížit dobu generování textury pod **40ms**, aby byl program plynulý. Následující tabulka vypisuje potřebný čas v závislosti na velikosti textury.
Čas generování:

- Vertikální řezy - náročnější:

Rozlišení textury	**Čas generování**
512px * 178px	30-40ms
512px * 378px	50-70ms

Tabulka 7.1: Čas generování textury - vertikální

- Horizontální řezy:

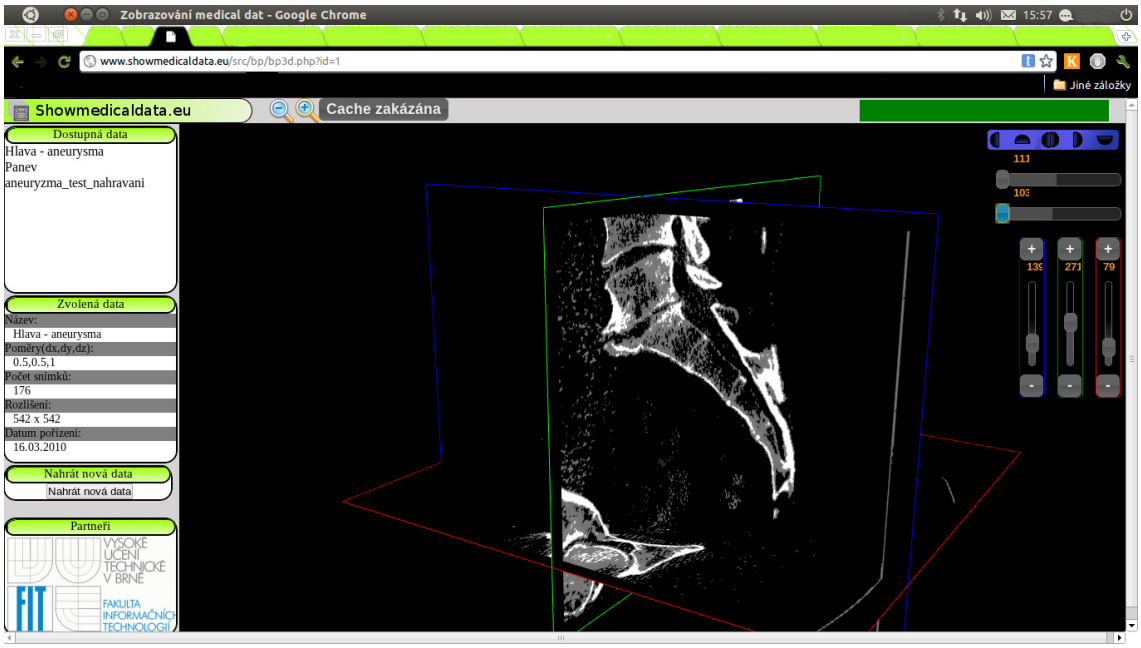
Rozlišení textury	**Čas generování**
512px * 512px	40-60ms

Tabulka 7.2: Čas generování textury - horizontální

Paměťová náročnost:

Rozlišení textury	Obsazení paměti (cca)
512px * 512px *178px	300MB
512px * 512px * 378px	650MB

Tabulka 7.3: Paměťová náročnost programu



Obrázek 7.1: Výsledná aplikace

I přesto, že rozlišení 512px * 512px se počítá déle než 40ms, tedy jeho pohyb není plynulý, je toto rozlišení ještě přípustné. Doporučujeme však rozlišení pod touto hranicí.

Doporučená konfigurace počítače: Aplikace sama dvě jádra nevyužije, ale jedno

CPU	2x2GHz
RAM	4GB
Grafická karta s akcelerací	
Síť	alespoň 8Mbit

Tabulka 7.4: Doporučená konfigurace PC

jádro zabere spolehlivě. Proto je třeba druhé jádro pro ostatní aplikace v systému.

Aplikace zobrazuje medicínská data a umožňuje nahrávání nových dat na server. Neumožňuje správu jednotlivých účtů, volume rendering, komunikaci a interaktivní spolupráci nad stejnými daty mezi uživateli(například lékaři). Pevně věřím v další vývoj této aplikace a všechny tyto věci bych velmi rád zahrnul do budoucího vývoje.

Kapitola 8

Závěr

Cílem práce bylo zjistit možnosti zobrazení akcelerované 3D grafiky v okně webového prohlížeče. Pro demonstraci technologií jsem zkusil zrealizovat aplikaci pro multiplanární zobrazení medicínských dat. Aplikace využívá akceleraci grafické karty a spolehlivě běží v prohlížečích podporujících technologii WebGL. Tím bylo zobrazení 3D scény ve webovém prohlížeči otestováno a zadání bakalářské práce splněno.

Zobrazení 3D scény v okně webového prohlížeče má dle mého názoru velkou budoucnost. Jedinou slabinou těchto aplikací je pomalá interpretace jazyka Javascript, ač o proti dřívějším dobám je jeho současná rychlost enormní. Alespoň co se prohlížečů založených na V8 (javascriptový engine) týče. Již nyní ale není problém psát méně náročné aplikace. Dostupných frameworků, pro snažší práci s grafikou je mnoho. Jsou podrobněji popsány v sekci 4.2.

Realizovaná aplikace pro multiplanární zobrazení medicínských dat nejvíce tratí právě na generování textur, protože je nutné zpracovávat velmi rozměrná prostorová pole. Tato slabina bude doufejme časem zcela eliminována. Rotace či posun objektu jsou akcelerované grafickou kartou. WebGL funguje zcela korektně. V dubnu 2011 byla vydána jeho finální specifikace.

Další alternativou by mohlo být propojení technologie Native Client(který by prováděl časově náročné výpočty v nativním kódu) a zbytku naší aplikace. Tato alternativa stojí za uvážení také. Native Client byl ale v době vývoje aplikace v počátcích svého vývoje, a proto tato alternativa byla zavrhnuta a odložena pro pozdější zkoumání.

Aplikace je lehce rozšiřitelná, ačkoli není plně objektová. Její hlavní řídící funkce je popsána strukturovaně a ke svému běhu používá dostupné objekty. Jako nevýhoda by se mohlo zdát přílišné propojení objektů způsobené globálními proměnnými, které umožňují přístup do O3D API. Zvyšuje se ale rychlost zpracování a to je pro nás přednější.

Literatura

[1] drishti. [cit. 2011-04-20], volume Exploration and Presentation Tool.
URL http://anusf.anu.edu.au/Vizlab/drishti/

[2] Seg3D. [cit. 2011-04-22], volumetric Image Segmentation and Visualization. Scientific Computing and Imaging Institute (SCI).
URL http://www.seg3d.org

[3] Native Client SDK. [cit. 2011-05-02].
URL http://code.google.com/intl/cs-CZ/chrome/nativeclient/

[4] O3D. [cit. 2011-05-03], o3D, O3D with WebGL.
URL http://code.google.com/intl/cs-CZ/apis/o3d/

[5] Ambierra: CopperLicht. [cit. 2011-04-22], fast WebGL JavaScript 3D Engine.
URL http://www.ambiera.com/copperlicht/

[6] Bartoň, R.; Kršek, P.; Španěl, M.; aj.: Virtual Colaborative Environment. 2009[cit. 2011-05-07], uPGM-PGMED, FIT VUT Brno, CESNET.
URL http://zdislava.fit.vutbr.cz/vce-public/about

[7] BWH and 3D Slicer contributors: 3DSlicer. 2011[cit. 2011-05-01], a multi-platform, free and open source software package for visualization and medical image computing.
URL www.slicer.org

[8] Drebin, R. A.; Carpenter, L.; Hanrahan, P.: Volume rendering. *SIGGRAPH Comput. Graph.*, ročník 22, June 1988: s. 65–74, ISSN 0097-8930, doi:http://doi.acm.org/10.1145/378456.378484.
URL http://doi.acm.org/10.1145/378456.378484

[9] E.R. Bachmann, P. M.: Viewport Transformations. Leden 2003.
URL http://www.movesinstitute.org/~mcdowell/mv4202/notes/lect8.pdf

[10] Herbig, M.: Srovnání výkonu internetových prohlížečů. January 2010.
URL http://www.swmag.cz/587/srovnani-vykonu-internetovych-prohlizecu/

[11] Kazík, I. J.: *Vizualizace objemových dat pomocí volume renderingu*. Diplomová práce, Fakulta informačních technologií, VUT, Brno, Česká republika, 2008.

[12] Khronos Group: 2D Texture Volume Rendering. 1997–2011[cit. 2011-05-02].
URL http://www.opengl.org/resources/code/samples/advanced/advanced97/notes/node182.html

[13] Khronos Group: Collada. 2011 [cit. 2011-05-01], collada - 3D Asset Exchange Schema.
URL http://www.khronos.org/collada/

[14] Michal Španěl, V. B.: Obrazové segmentační techniky. 2006 [cit. 2011-05-08], přehled existujících metod.
URL http://www.fit.vutbr.cz/~spanel/segmentace/

[15] Phillipe Lacroute, M. L.: Fast Volume Rendering Using a Shear-Warp Factorization of the Viewing Transformation. July 1994: s. 451–458.

[16] Ross Moore, N. D.: BioImage Suite. 2002[cit. 2011-04-29].
URL http://www.bioimagesuite.org/doc/node4.html

[17] Rosset, A.: OsiriX Imaging software. 2003-2011 [cit. 2011-04-28], advanced Open-Source PACS Workstation DICOM Viewer.
URL http://www.osirix-viewer.com/

[18] Sandia Corporation, K.: ParaView. [cit. 2011-04-25].
URL http://www.paraview.org/

[19] Tatiana Al-Chueyr, T. F. d. M., Paulo Henrique Junqueira Amorim: InVesalius @ONLINE. [cit. 2011-04-22, mecical Imaging Public Software.
URL http://svn.softwarepublico.gov.br/trac/invesalius

Příloha A

Obsah CD

Optické médium, které je k práci přiloženo, obsahuje následující soubory a složky:

- **O3DAPI** - framework O3D
- **Showmedicaldata** - aplikace pro multiplanární zobrazení
- **Showmedicaldata/css** - veškerá potřebná grafika pro stylizaci aplikace včetně souborů se styly
- **Showmedicaldata/jquery-ui-1.8.7.custom** - framework jQuery
- **Showmedicaldata/js** - obsahuje objekty pro vytvoření objektů, nahrávání dat, stahování dat ze serveru, obsluhu klávesnice, obsluhu myši, manipulaci s objekty, projekci bodu z 3D do roviny obrazovky, generování a vykreslení textur
- **Showmedicaldata/medicalData** - obsahuje testovací data (snímky)
- **Showmedicaldata/modules** - knihovny třetích stran
- **Showmedicaldata/bp3d.php** - hlavní řídící skript programu
- **Showmedicaldata/loadData.php** - nahrává data na server
- **Showmedicaldata/sendData.php** - posílá data klientům
- **Showmedicaldata/uploadDialog.php** - nahrávací dialog nových dat na server

Příloha B

Manuál

- **Zprovoznění aplikace na serveru:**
  Nahrání obsahu na server nebo
  1. Stažení aktuálního O3DAPI - to umístit do samostatné složky.
  2. Složku s programem umístit do stejné složky jako jsme umístili složku O3DAPI.
  3. Vytvořit soubor index.php a do něj vložit:

```
<?php
   header( 'Location: http://vase_domena/Showmedicaldata/bp3d.php' ) ;
?>
```

- **Spuštění aplikace:**
  Přistoupení na adresu www.showmedicaldata.eu .

- **Klávesové zkratky:**
  levé tlačítko myši + pohyb →rotace objektu
  levý control + levé tlačítko myši + pohyb →pohyb s řezy
  levý shift + levé tlačítko myši + pohyb →natáčení kamery

- **Densitní okénko:**
  Dva horizontální slidery v pravém horním rohu zobrazovací plochy.

Příloha C

Plakát